IIIB.Tech. I SEM **Regulation: R20**

Laboratory Manual

For the course of

MACHINE LEARNING LAB

Branch: COMPUTER SCIENCE AND ENGINEERING
CSM ,CAI& AIML

VIGNAN'S LARA
INSTITUTE OF TECHNOLOGY & SCIENCE
(AUTONOMOUS)
Approved by AICTE New Delhi & Affiliated to JNTUK Kakinada
Accredited by NAAC 'A+' and NBA | ISO 9001 : 2015
Vadlamudi - 522 213, Guntur District

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY: KAKINADA
KAKINADA – 533 003, Andhra Pradesh, India

DEPARTMENT OF CSE - ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

III B Tech I Sem		L	T	P	C
		0	0	3	1.5
MACHINE LEARNING LAB					

Course Objectives:
This course will enable students to learn and understand different Data sets in implementing the machine learning algorithms.

Course Outcomes (Cos): At the end of the course, student will be able to

- Implement procedures for the machine learning algorithms
- Design and Develop Python programs for various Learning algorithms
- Apply appropriate data sets to the Machine Learning algorithms
- Develop Machine Learning algorithms to solve real world problems

Requirements: Develop the following program using Anaconda/ Jupiter/ Spider and evaluate ML models.

Experiment-1:
Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

Experiment-2:
For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

Experiment-3:
Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Experiment-4:
Exercises to solve the real-world problems using the following machine learning methods: a) Linear Regression b) Logistic Regression c) Binary Classifier

Experiment-5: Develop a program for Bias, Variance, Remove duplicates , Cross Validation
Experiment-6: Write a program to implement Categorical Encoding, One-hot Encoding

Experiment-7:
Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

Experiment-8:
Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions.

Experiment-9: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY: KAKINADA
KAKINADA – 533 003, Andhra Pradesh, India

DEPARTMENT OF CSE - ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

Experiment-10:
Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

Experiment-11: Apply EM algorithm to cluster a Heart Disease Data Set. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

Experiment-12: Exploratory Data Analysis for Classification using Pandas or Matplotlib.

Experiment-13:
Write a Python program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set

Experiment-14:
Write a program to Implement Support Vector Machines and Principle Component Analysis

Experiment-15:
Write a program to Implement Principle Component Analysis

INDEX

Exp.No	Name of the Experiment	Page.No
1.	Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.	1
2.	For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of allhypotheses consistent with the training examples.	2 -7
3.	Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample .	8 - 11
4.	Exercises to solve the real-world problems using the following machine learning methods: a) Linear Regression b) Logistic Regression c) Binary Classifier	12 - 15
5.	Develop a program for Bias, Variance, Remove duplicates , Cross Validation	16
6.	Write a program to implement Categorical Encoding, One-hot Encoding	17 - 18
7.	Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.	19 - 20
8.	Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions.	21 - 22
9.	Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.	23 - 24
10.	Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.	25 - 26
11.	Apply EM algorithm to cluster a Heart Disease Data Set. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.	27 - 29

12.	Exploratory Data Analysis for Classification using Pandas or Matplotlib	30 - 31
13.	Write a Python program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set	32 - 36
14.	Write a program to Implement Support Vector Machines and Principle Component Analysis	37 - 38
15.	Write a program to Implement Principle Component Analysis	39 - 40

1. **AIM: Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.**

Source Code:

```
import csv

with open ('tennis.csv', 'r') as
   f:reader = csv. reader(f)
   your_list = list(reader)

h = [['0', '0', '0', '0', '0', '0']]

for i in your_list:
   print(i)
   if  i[-1] =="True":
      j = 0
      for x in i:
         if  x != "True":
            if  x != h[0][j] and h[0][j]  ==
               '0':
            h[0][j] = x
            elif  x != h[0][j] and h[0][j] !=
               '0':
            h[0][j] = '?'
            else:
               pass
         j = j + 1
print("Most specific hypothesis is")
print(h)
```

Output

'Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', True

'Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', True

'Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', False

'Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', True

Maximally Specific set

[['Sunny', 'Warm', '?', 'Strong', '?', '?']]

2. AIM: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

Source Code:

```
class Holder:
    factors= {}          #Initialize an empty dictionary
    attributes = ()      #declaration of dictionaries parameters with an arbitrary length

    '''
    Constructor of class Holder holding two parameters,
    self refers to the instance of the class
    '''
    def init (self, attr): #
        self. Attributes =
        attr for i in attr:
            self. Factors[i]=[]

    def add_values (self, factor,
        values):self.
        factors[factor]=values


class CandidateElimination:
    Positive= {} #Initialize positive empty dictionary
    Negative={} #Initialize negative empty dictionary


    def-init-(self, data, fact):
        self.num_factors = len(data
        [0][0])self.factors = fact. Factors
        self.attr = fact.attributes
        self.dataset = data


    def run_algorithm(self):
        '''
        Initialize the specific and general boundaries, and loop the dataset against the
algorithm
        '''
        G = self.initializeG()
        S = self.initializeS()


        '''
```

```
        Programmatically populate list in the iterating variable trial_set
        '''
        count=0
        for trial_set in self.dataset:
            if self.is_positive(trial_set): #if trial set/example consists of positive examples

                        G = self.remove_inconsistent_G(G,trial_set[0]) #remove
                        inconsitent data fromthe general boundary


                S_new = S[:] #initialize the dictionary with no key-value pair
                print (S_new)
                for s in S:
                    if not self.consistent(s,trial_set[0]):
                        S_new.remove(s)
                        generalization = self.generalize_inconsistent_S(s,trial_set[0])
                        generalization = self.get_general(generalization,G)
                        if generalization:
                            S_new.append(generalization)
                    S = S_new[:]
                    S = self.remove_more_general(S)
                print(S)

            else:  #if it is negative


                S = self.remove_inconsistent_S(S,trial_set[0]) #remove inconsitent data from
the specific boundary
                G_new = G[:] #initialize the dictionary with no key-value pair (dataset can
take any value)
                print (G_new)
                for g in G:
                  if self.consistent(g,trial_set[0]):
                        G_new.remove(g)
                        specializations = self.specialize_inconsistent_G(g,trial_set[0])
                        specializationss = self.get_specific(specializations,S)
                        if specializations != []:
                            G_new += specializationss
                    G = G_new[:]
                    G = self.remove_more_specific(G)
                print(G)

        print (S)
        print (G)
```

```
def initializeS(self):
    ''' Initialize the specific boundary '''
    S = tuple(['-' for factor in range(self.num_factors)]) #6 constraints in the vector
    return [S]


def initializeG(self):
    ''' Initialize the general boundary '''
    G = tuple(['?' for factor in range(self.num_factors)]) # 6 constraints in the vector
    return [G]

def is_positive(self,trial_set):
    ''' Check if a given training trial_set is positive '''
    if trial_set[1] == 'Y':
        return True
    elif trial_set[1] == 'N':
        return False
    else:
        raise TypeError("invalid target value")

def match_factor(self,value1,value2):
    ''' Check for the factors values match,
        necessary while checking the consistency of
        training trial_set with the hypothesis '''
    if value1 == '?' or value2 == '?':
        return True
    elif value1 == value2 :
        return True
    return False
def consistent(self,hypothesis,instance):
    ''' Check whether the instance is part of the hypothesis '''
    for i,factor in enumerate(hypothesis):
        if not self.match_factor(factor,instance[i]):
            return False
    return True


def remove_inconsistent_G(self,hypotheses,instance):
    ''' For a positive trial_set, the hypotheses in G
        inconsistent with it should be removed '''
    G_new = hypotheses[:]


    for g in hypotheses:
        if not self.consistent(g,instance):
            G_new.remove(g)
    return G_new
```

```
def remove_inconsistent_S(self,hypotheses,instance):'''
For a negative trial_set, the hypotheses in S
        inconsistent with it should be removed '''
    S_new = hypotheses[:]
    for s in hypotheses:
        if self.consistent(s,instance):
            S_new.remove(s)
    return S_new


  def remove_more_general(self,hypotheses):
    ''' After generalizing S for a positive trial_set, the hypothesis in S
     general than others in S should be removed '''
    S_new = hypotheses[:]
    for old in hypotheses:
        for new in S_new:
            if old!=new and self.more_general(new,old):
                S_new.remove[new]
    return S_new


  def remove_more_specific(self,hypotheses):
    ''' After specializing G for a negative trial_set, the hypothesis in G
    specific than others in G should be removed '''
    G_new = hypotheses[:]
     for old in hypotheses:
       for new in G_new:
            if old!=new and self.more_specific(new,old):
                G_new.remove[new]
  return G_new def generalize_inconsistent_S(self,hypothesis,instance):
    ''' When a inconsistent hypothesis for positive trial_set is seen in the specific
boundary S,
        it should be generalized to be consistent with the trial_set ... we will get one
hypothesis'''
    hypo = list(hypothesis) # convert tuple to list for mutability
    for i,factor in enumerate(hypo):
        if factor == '-':
            hypo[i] = instance[i]
        elif not self.match_factor(factor,instance[i]):
            hypo[i] = '?'
    generalization = tuple(hypo) # convert list back to tuple for immutability
    return generalization
```

```
def specialize_inconsistent_G(self,hypothesis,instance):
    ''' When a inconsistent hypothesis for negative trial_set is seen in the general boundary G
        should be specialized to be consistent with the trial_set.. we will get a set of hypotheses '''
    specializations = []
    hypo = list(hypothesis) # convert tuple to list for mutability
    for i,factor in enumerate(hypo):
        if factor == '?':
            values = self.factors[self.attr[i]]
            for j in values:
                if  instance[i] !=
                    j:hyp=hypo[:]
                    hyp[i]=j
                    hyp=tuple(hyp) # convert list back to tuple for immutability
                    specializations.append(hyp)
    return specializations
def get_general(self,generalization,G):
    ''' Checks if there is more general hypothesis in G
        for a generalization of inconsistent hypothesis in S
        in case of positive trial_set and returns valid generalization '''

    for g in G:
        if self.more_general(g,generalization):
            return generalization
    return None
def get_specific(self,specializations,S):
    ''' Checks if there is more specific hypothesis in S
        for each of hypothesis in specializations of an
        inconsistent hypothesis in G in case of negative trial_set
        and return the valid specializations'''
    valid_specializations = []
    for hypo in specializations:
        for s in S:
            if self.more_specific(s,hypo) or s==self.initializeS()[0]:
                valid_specializations.append(hypo)
    return valid_specializations

def exists_general(self,hypothesis,G):
```

```
    '''Used to check if there exists a more general hypothesis in
       general boundary for version space''' def
       exists_general(self,hypothesis,G):
    '''Used to check if there exists a more general hypothesis in
       general boundary for version space'''

    for g in G:
       if self.more_general(g,hypothesis):
          return True
    return False

def exists_specific(self,hypothesis,S):
    '''Used to check if there exists a more specific hypothesis in
       general boundary for version space'''

    for s in S:
       if self.more_specific(s,hypothesis):
          return True
    return False

def more_general(self,hyp1,hyp2):
    ''' Check whether hyp1 is more general than hyp2 '''
    hyp = zip(hyp1,hyp2)
    for i,j in hyp:
       if i == '?':
          continue
       elif j == '?':
          if i != '?':
             return False
       elif i != j:
          return False
       else:
          continue
    return True
def more_specific(self,hyp1,hyp2):
    ''' hyp1 more specific than hyp2 is
       equivalent to hyp2 being more general than hyp1 '''
    return self.more_general(hyp2,hyp1)
```

```
dataset=[(('sunny','warm','normal','strong','warm','same'),'Y'),(('sunny','warm','high','stron
g','warm','same'),'Y'),(('rainy','cold','high','strong','warm','change'),'N'),(('sunny','warm','hi
gh','strong','cool','change'),'Y')]
attributes =('Sky','Temp','Humidity','Wind','Water','Forecast')
f = Holder(attributes)
f.add_values('Sky',('sunny','rainy','cloudy')) #sky can be sunny rainy or cloudy
f.add_values('Temp',('cold','warm')) #Temp can be sunny cold or warm
f.add_values('Humidity',('normal','high')) #Humidity can be normal or high
f.add_values('Wind',('weak','strong')) #wind can be weak or strong
f.add_values('Water',('warm','cold')) #water can be warm or cold
f.add_values('Forecast',('same','change')) #Forecast can be same or change
a = Candidate Elimination(dataset,f) #pass the dataset to the algorithm class and call
therun algoritm method
a.run_algorithm()
```

Output

```
[('sunny', 'warm', 'normal', 'strong', 'warm', 'same')]
[('sunny', 'warm', 'normal', 'strong', 'warm', 'same')]
[('sunny', 'warm', '?', 'strong', 'warm', 'same')]
[('?', '?', '?', '?', '?', '?')]
[('sunny', '?', '?', '?', '?', '?'), ('?', 'warm', '?', '?', '?', '?'), ('?', '?', '?', '?', '?', 'same')]
[('sunny', 'warm', '?', 'strong', 'warm', 'same')]
[('sunny', 'warm', '?', 'strong', '?', '?')]
[('sunny', 'warm', '?', 'strong', '?', '?')]
[('sunny', '?', '?', '?', '?', '?'), ('?', 'warm', '?', '?', '?', '?')]
```

3.AIM: Write a program to demonstrate the working of the decision tree based ID3algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Source Code:

```
import numpy as npimport math
from data_loader import read_data


class Node:
    def init (self, attribute): self.attribute =
        attributeself.children = [] self.answer = ""


    def__str__(self): return self.attribute


def subtables(data, col, delete):dict = {}
    items = np.unique(data[:, col])
    count = np.zeros((items.shape[0], 1), dtype=np.int32)for x in
    range(items.shape[0]):
        for y in range(data.shape[0]):if data[y, col] ==
            items[x]:
                count[x] += 1


    for x in range(items.shape[0]):
        dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype="|S32")pos = 0
        for y in range(data.shape[0]):if data[y, col] ==
            items[x]:
                dict[items[x]][pos] = data[y]pos += 1

        if delete:
            dict[items[x]] = np.delete(dict[items[x]], col, 1)return items, dict
def entropy(S):
items = np.unique(S)
if items.size == 1:
return 0


                        counts = np.zeros((items.shape[0], 1))
                        sums = 0

                        for x in range(items.shape[0]):


                            counts[x] = sum(S == items[x]) / (S.size * 1.0)
                        for count in counts:
```

```
        sums += -1 * count * math.log(count, 2)
    return sums

def gain_ratio(data, col):
    items, dict = subtables(data, col, delete=False)

    total_size = data.shape[0]
    entropies = np.zeros((items.shape[0], 1))
    intrinsic = np.zeros((items.shape[0], 1))
    for x in range(items.shape[0]):
        ratio = dict[items[x]].shape[0]/(total_size * 1.0)
        entropies[x] = ratio * entropy(dict[items[x]][:, -1])
        intrinsic[x] = ratio * math.log(ratio, 2)

    total_entropy = entropy(data[:, -1])
    iv = -1 * sum(intrinsic)

    for x in range(entropies.shape[0]):
        total_entropy -= entropies[x]

    return total_entropy / iv


def create_node(data, metadata):
    if (np.unique(data[:, -1])).shape[0] == 1:
        node = Node("")
        node.answer = np.unique(data[:, -1])[0]
        return node

    gains = np.zeros((data.shape[1] - 1, 1))
    for col in range(data.shape[1] - 1):
        gains[col] = gain_ratio(data, col)

    split = np.argmax(gains)

    node = Node(metadata[split])
    metadata = np.delete(metadata, split, 0)

    items, dict = subtables(data, split, delete=True)
```

```
    for x in range(items.shape[0]):
        child = create_node(dict[items[x]], metadata)
        node.children.append((items[x], child))
    return node
def empty(size):
    s = ""
    for x in range(size):
        s += " "
    return s

def print_tree(node, level):
    if node.answer != "":
        print(empty(level), node.answer)
        return
    print(empty(level), node.attribute)
    for value, n in node.children:
        print(empty(level + 1), value)
        print_tree(n, level + 2)



metadata, traindata = read_data("tennis.csv")
data = np.array(traindata)
node = create_node(data, metadata)
print_tree(node, 0)
```

Data_loader.py

```
import csv
def read_data(filename):
    with open(filename, 'r') as csvfile:
        datareader = csv.reader(csvfile, delimiter=',')
        headers = next(datareader)
        metadata = []
        traindata = []
        for name in headers:
            metadata.append(name)
    for row in datareader: traindata.append(row)
     return (metadata, traindata)
```

Tennis.csv

outlook, temperature, humidity,
wind, answer sunny, hot, high,
weak, no sunny, hot ,high, strong
,no overcast, hot ,high, weak, yes
rain, mild, high, weak, yes rain,
cool ,normal, weak, yes rain, cool,
normal, strong, no overcast, cool,
normal, strong, yes sunny, mild,
high, weak ,no sunny, cool ,normal
,weak, yes rain, mild ,normal, weak
,yes sunny ,mild ,normal ,strong,
yes overcast t, mild ,high, strong,
yes overcast ,hot ,normal, weak,
yes rain ,mild ,high ,strong, no

Output

```
outlook
  overcast
    b 'yes'
  rain
    wind
      b'strong'
        b'no'
      b'weak'
        b'yes'
    sunny
    humidity
      b'high'
        b'no'
      b'normal'
        b'yes
```

4.AIM: Exercises to solve the real-world problems using the following machine learning methods:

a)Linear Regression

b) Logistic Regression

c) Binary Classifier.

a) Linear Regression

Source Code:

```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X,axis=0) # maximum of X array longitudinally
y = y/100

#Sigmoid Function
def sigmoid (x):
    return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)

#Variable initialization
epoch=7000 #Setting training iterations
lr=0.1 #Setting learning rate
inputlayer_neurons = 2 #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer
#weight and bias initialization
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
#draws a random range of numbers uniformly of dim x*y
for i in range(epoch):
```

```
#Forward Propogation
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+ bout
    output = sigmoid(outinp)

#Backpropagation
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO* outgrad
    EH = d_output.dot(wout.T)
    hiddengrad = derivatives_sigmoid(hlayer_act)#how much hidden layer wts

contributed to error
    d_hiddenlayer = EH * hiddengrad
    wout += hlayer_act.T.dot(d_output) *lr# dotproduct of nextlayererror and
currentlayerop
   # bout += np.sum(d_output, axis=0,keepdims=True) *lr
    wh += X.T.dot(d_hiddenlayer) *lr
    #bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

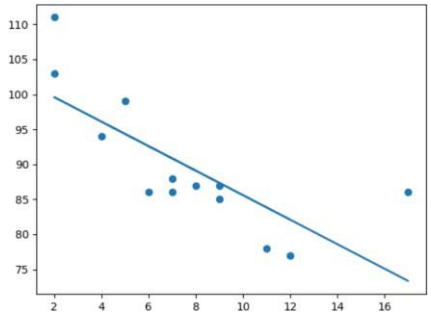
output

```
Input:
 [[ 0.66666667 1. ]
 [ 0.33333333 0.55555556]
 [ 1.     0.66666667]]
Actual Output:[[ 0.92]
 [ 0.86]
 [ 0.89]]
Predicted Output:[[ 0.89559591]
 [ 0.88142069]
 [ 0.8928407 ]]
```

a) Logistic RegressionSource Code:

```
import matplotlib.pyplot as pltfrom

scipy import stats


x = [5,7,8,7,2,17,2,9,4,11,12,9,6]

y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

slope, intercept, r, p, std_err = stats.linregress(x, y)def myfunc(x):

    return slope * x + intercept

mymodel = list(map(myfunc, x))

plt.scatter(x, y)

plt.plot(x, mymodel)

plt.show()
```

Output:



b) Binary Classifier Source

Code:

```
import sklearn as sk
import pandas as pd
import pandas as pd
import os
rom sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier


os.chdir('/Users/stevenhurwitt/Documents/Blog/Classification')
heart = pd.read_csv('SAHeart.csv', sep=',', header=0)
heart.head()


y = heart.iloc[:,9]


X = heart.iloc[:,:9]
vowel_train = pd.read_csv('vowel.train.csv', sep=',', header=0)
vowel_test = pd.read_csv('vowel.test.csv', sep=',', header=0)


vowel_train.head()
```

```
y_tr = vowel_train.iloc[:,0]
X_tr = vowel_train.iloc[:,1:]

y_test = vowel_test.iloc[:,0]
X_test = vowel_test.iloc[:,1:]

   NN = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2),
random_state=1)NN.fit(X, y)
NN.predict(X.iloc[460:,:])
round(NN.score(X,y), 4)

SVM = svm.SVC(decision_function_shape="ovo").fit(X_tr, y_tr)
SVM.predict(X_test)
round(SVM.score(X_test, y_test), 4)

RF = RandomForestClassifier(n_estimators=1000, max_depth=10, random_state=0).fit(X_tr,
y_tr)
RF.predict(X_test)
round(RF.score(X_test, y_test), 4)

NN = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(150, 10),
random_state=1).fit(X_tr, y_tr)
NN.predict(X_test)
round(NN.score(X_test, y_test), 4)
```

OutPut:

	y	x.1	x.2	x.3	x.4	x.5	x.6	x.7	x.8	x.9	x.10
0	1	-3.639	0.418	-0.670	1.779	-0.168	1.627	-0.388	0.529	-0.874	-0.814
1	2	-3.327	0.496	-0.694	1.365	-0.265	1.933	-0.363	0.510	-0.621	-0.488
2	3	-2.120	0.894	-1.576	0.147	-0.707	1.559	-0.579	0.676	-0.809	-0.049
3	4	-2.287	1.809	-1.498	1.012	-1.053	1.060	-0.567	0.235	-0.091	-0.795
4	5	-2.598	1.938	-0.846	1.062	-1.633	0.764	0.394	-0.150	0.277	-0.396

5AIM: Develop a program for Bias, Variance, remove duplicates, Cross Validation

Source Code:

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from mlxtend.evaluate import bias_variance_decomp
# load dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv'
dataframe = read_csv(url, header=None)
# separate into inputs and outputs
data = dataframe.values
X, y = data[:, :-1], data[:, -1]
# split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)
# define the model
model = LinearRegression()
# estimate bias and variance
mse, bias, var = bias_variance_decomp(model, X_train, y_train, X_test, y_test, loss='mse',
num_rounds=200, random_seed=1)
# summarize results
print('MSE: %.3f' % mse)
print('Bias: %.3f' % bias)
print('Variance: %.3f' % var)
```

Output:

```
MSE: 22.487
Bias: 20.726
Variance: 1.761
```

6.AIM: Write a program to implement Categorical Encoding, One-hot Encoding.

Source Code:

 # Program for demonstration of one hot encoding

```
# import libraries


import numpy as np
import pandas as pd


# import the data required
data = pd.read_csv('employee_data.csv')
print(data.head())


print(data['Gender'].unique())
print(data['Remarks'].unique())
```

Output:

```
array(['Male', 'Female'], dtype=object)
array(['Nice', 'Good', 'Great'], dtype=object)
```

 # Program for demonstration of Categorical encoding

```
# importing libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder

# Retrieving data
data = pd.read_csv('Employee_data.csv')

# Converting type of columns to category
data['Gender'] = data['Gender'].astype('category')
data['Remarks'] = data['Remarks'].astype('category')

# Assigning numerical values and storing it in another columns
data['Gen_new'] = data['Gender'].cat.codes
data['Rem_new'] = data['Remarks'].cat.codes
```

```
# Create an instance of One-hot-encoderenc = OneHotEncoder()

 # Passing encoded columns

enc_data = pd.DataFrame(enc.fit_transform(

data[['Gen_new', 'Rem_new']]).toarray())#

Merge with main

New_df = data.join(enc_data)

print(New_df)
```

Output:

	Employee_Id	Gender	Remarks	Gen_new	Rem_new	0	1	2	3	4
0	45	Male	Nice	1	2	0.0	1.0	0.0	0.0	1.0
1	78	Female	Good	0	0	1.0	0.0	1.0	0.0	0.0
2	56	Female	Great	0	1	1.0	0.0	0.0	1.0	0.0
3	12	Male	Great	1	1	0.0	1.0	0.0	1.0	0.0
4	7	Female	Nice	0	2	1.0	0.0	0.0	0.0	1.0
5	68	Female	Great	0	1	1.0	0.0	0.0	1.0	0.0
6	23	Male	Good	1	0	0.0	1.0	1.0	0.0	0.0
7	45	Female	Nice	0	2	1.0	0.0	0.0	0.0	1.0
8	89	Male	Great	1	1	0.0	1.0	0.0	1.0	0.0
9	75	Female	Nice	0	2	1.0	0.0	0.0	0.0	1.0
10	47	Female	Good	0	0	1.0	0.0	1.0	0.0	0.0
11	62	Male	Nice	1	2	0.0	1.0	0.0	0.0	1.0

7.AIM: Build an Artificial Neural Network by implementing the Back propagation algorithmand test the same using appropriate datasets.

Source Code:

```
Import numpy as np
X =np.array(([2,9],[1,5],[3,6]),dtype=float)
y=np.array(([92],[86],[89]),dtype=float)
X=X/np.amax(X,axis=0)    #maximumofXarraylongitudinallyy= y/100

#Sigmoid Functiondefsigmoid(x):
return1/(1+np.exp(-x))

#DerivativeofSigmoidFunctiondefderivatives_sigmoid(x):
returnx* (1-x)
#Variableinitialization
epoch=7000#Settingtrainingiterationslr=0.1#Settinglearning rate
inputlayer_neurons = 2 #number of features in data
sethiddenlayer_neurons=3#numberofhiddenlayersneuronsoutput_neurons = 1 #number of neurons at output layer#weightand biasinitialization

wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))bout=np.random.uniform(size=(1,output_neurons))
#drawsarandomrangeofnumbersuniformlyofdimx*yforiin range(epoch):

#Forward Propogationhinp1=np.dot(X,wh)hinp=hinp1 + bhhlayer_act=sigmoid(hinp)
outinp1=np.dot(hlayer_act,wout)outinp=outinp1+ bout
output=sigmoid(outinp)
```

```
#BackpropagationEO=y-output
outgrad=derivatives_sigmoid(output)d_output= EO* outgrad
EH=d_output.dot(wout.T)
hiddengrad=derivatives_sigmoid(hlayer_act)#how muchhiddenlayerwtscontributedto error

d_hiddenlayer=EH*hiddengrad
wout+=hlayer_act.T.dot(d_output)*lr#dotproductofnextlayererrorandcurrentlayerop
# bout += np.sum(d_output, axis=0,keepdims=True) *lrwh+=X.T.dot(d_hiddenlayer) *lr
#bh+=np.sum(d_hiddenlayer,axis=0,keepdims=True)*lrprint("Input:\n"+ str(X))
print("Actual Output: \n" + str(y))print("PredictedOutput:\n",output)
```

Output:

```
Input:
[[ 0.666666671. ]
[0.333333330.55555556]
[1.        0.66666667]]
Actual Output:[[ 0.92]
[0.86]
[0.89]]
Predicted Output: [[0.89559591]
[0.88142069]
[0.8928407]]
```

6.AIM: Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set.Print both correct and wrong predictions.

Source Code:

```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

# Read dataset to pandas dataframe
dataset = pd.read_csv("9-dataset.csv", names=names)
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1]
print(X.head())
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.10)


classifier = KNeighborsClassifier(n_neighbors=5).fit(Xtrain, ytrain)
ypred = classifier.predict(Xtest)
i = 0
print ("\n----------")
print ('%-25s %-25s %-25s' % ('Original Label', 'Predicted Label', 'Correct/Wrong'))
print ("----------")
for label in ytest:
    print ('%-25s %-25s' % (label, ypred[i]), end="")
    if (label == ypred[i]):
        print (' %-25s' % ('Correct'))
    else:
        print (' %-25s' % ('Wrong'))
    i = i + 1
print ("----------")
print("\nConfusion Matrix:\n",metrics.confusion_matrix(ytest, ypred))
print ("----------")
print("\nClassification Report:\n",metrics.classification_report(ytest, ypred))
print ("----------")
print('Accuracy of the classifer is %0.2f' % metrics.accuracy_score(ytest,ypred))
print ("----------")
```

Output:

```
   sepal-length  sepal-width  petal-length  petal-width
0           5.1          3.5           1.4          0.2
1           4.9          3.0           1.4          0.2
2           4.7          3.2           1.3          0.2
3           4.6          3.1           1.5          0.2
4           5.0          3.6           1.4          0.2

-------------------------------------------------------------------------
Original Label            Predicted Label           Correct/Wrong
-------------------------------------------------------------------------
Iris-versicolor           Iris-versicolor           Correct
Iris-virginica            Iris-versicolor           Wrong
Iris-virginica            Iris-virginica            Correct
Iris-versicolor           Iris-versicolor           Correct
Iris-setosa               Iris-setosa               Correct
Iris-versicolor           Iris-versicolor           Correct
Iris-setosa               Iris-setosa               Correct
Iris-setosa               Iris-setosa               Correct
Iris-virginica            Iris-virginica            Correct
Iris-virginica            Iris-versicolor           Wrong
Iris-virginica            Iris-virginica            Correct
Iris-setosa               Iris-setosa               Correct
Iris-virginica            Iris-virginica            Correct
Iris-virginica            Iris-virginica            Correct
```

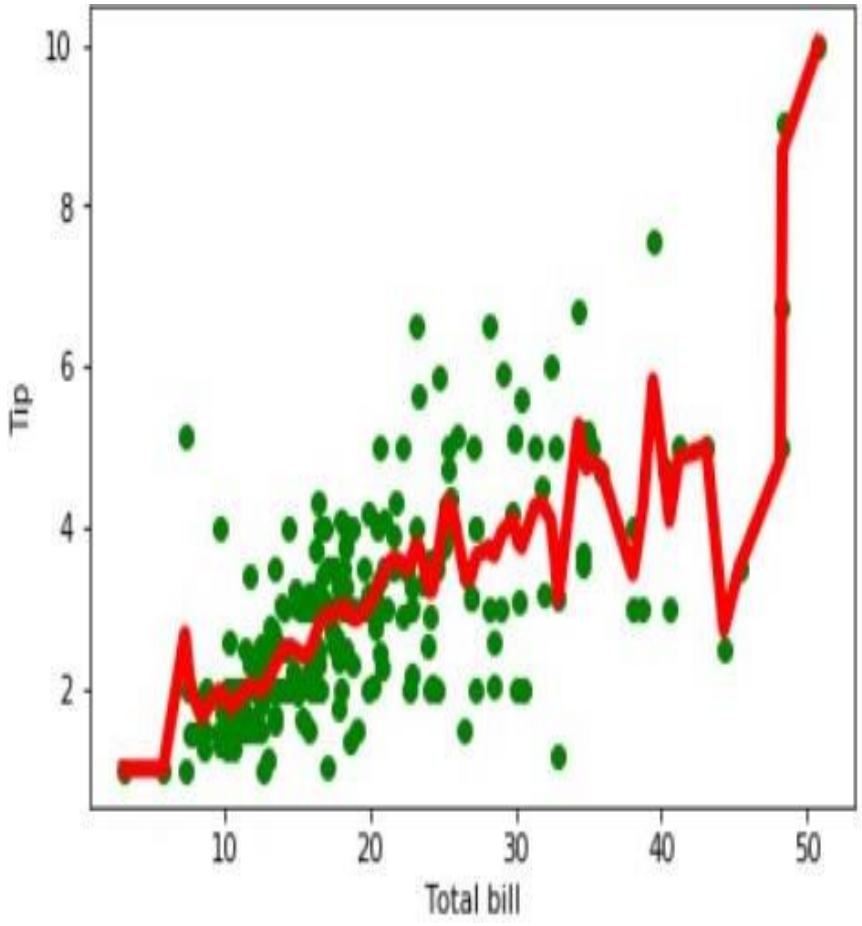
9.AIM: Implement the non-parametric Locally Weighted Regression algorithm in order to fitdata points. Select appropriate data set for your experiment and draw graphs.

Source Code:

```
import matplotlib. pyplot as plt
import pandas as pd
import numpy as np
def kernel(point, xmat, k):
m,n = np.shape(xmat)
weights = np.mat(np1.eye((m)))
for j in range(m):
diff = point - X[j]
weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
return weights
def localWeight(point, xmat, ymat, k):
wei = kernel(point,xmat,k)
W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
return W

def localWeightRegression(xmat, ymat, k):
m,n = np.shape(xmat)
ypred = np.zeros(m)
for i in range(m):
ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
return ypred
# load data points
data = pd.read_csv('10-dataset.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)
#preparing and add 1 in bill
mbill = np.mat(bill)
mtip = np.mat(tip)
```

```
m= np.shape(mbill)[1]
one = np.mat(np1.ones(m))
X = np.hstack((one.T,mbill.T))
#set k here
ypred = localWeightRegression(X,mtip,0.5)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='green')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt. show();
```

Output:



10.AIM: Assuming a set of documents that need to be classified, use the naïve Bayesian Classifiermodel to perform this task. Built-in Java classes/API can be used to write the program. Calculatethe accuracy, precision, and recall for your dataset.

Source Code:

```
import pandas as pd
msg=pd.read_csv('naivetext1.csv',names=['message','label'])
print('The dimensions of the dataset',msg.shape)msg['labelnum']=msg.label.map({'pos':1,'neg':0})
X=msg.messagey=msg.labelnumprint(X)
print(y)


#splittingthedataset intotrainandtestdata
fromsklearn.model_selectionimporttrain_test_splitxtrain,xtest,ytrain,ytest=train_test_split(X,y)
print(xtest.shape)
print(xtrain.shape)print(ytest.shape)print(ytrain.shape)


#outputofcountvectoriserisasparsematrix
fromsklearn.feature_extraction.textimportCountVectorizercount_vect= CountVectorizer()
xtrain_dtm=count_vect.fit_transform(xtrain)xtest_dtm=count_vect.transform(xtest)print(count_vect.get
_feature_names())

df=pd.DataFrame(xtrain_dtm.toarray(), columns=count_vect.get_feature_names())
print(df)#tabularrepresentation
print(xtrain_dtm)       #sparsematrixrepresentation

#TrainingNaiveBayes(NB)classifierontrainingdata.fromsklearn.naive_bayes importMultinomialNB
clf=MultinomialNB().fit(xtrain_dtm,ytrain)predicted=clf.predict(xtest_dtm)


#printing accuracy metricsfromsklearnimportmetricsprint('Accuracymetrics')
print('Accuracyoftheclassiferis',metrics.accuracy_score(ytest,predicted))
print('Confusionmatrix')
print(metrics.confusion_matrix(ytest,predicted))
print('Recall and Precison ')
print(metrics.recall_score(ytest,predicted))
print(metrics.precision_score(ytest,predicted))

'''docs_new=['Ilikethisplace','Mybossisnotmysaviour']
X_new_counts=count_vect.transform(docs_new)
predictednew =clf.predict(X_new_counts)
```

fordoc,categoryinzip(docs_new,predictednew):
print('%s->%s'%(doc, msg.labelnum[category]))"'

I love this sandwich,pos This is an amazing place,pos

Ifeelverygood aboutthesebeers,posThis ismybestwork,pos
Whatanawesomeview,pos

Idonotlike thisrestaurant,negIamtiredofthis stuff,neg

I can't deal with this,negHeismyswornenemy,negMyboss is horrible,neg
Thisisanawesomeplace,pos

Idonotlikethetasteofthis juice,negIloveto dance,pos
Iamsickand tiredofthisplace,negWhatagreatholiday,pos
Thatis abadlocalitytostay,neg

We will have good fun tomorrow,posIwenttomyenemy'shousetoday,neg

Output:

['about','am','amazing','an','and','awesome','beers','best','boss','can','deal',
'do','enemy','feel','fun','good','have','horrible','house','is','like','love','my',
'not','of','place','restaurant','sandwich','sick','stuff','these','this','tired','to',
'today','tomorrow','very','view','we', 'went','what','will','with','work']

About amamazing and awesome beers bestboss can...today

0 1 0 0 0 0 01 0 0 0 ... 0
1 0 0 0 0 0 0 0 1 0 0 ... 0
2 0 0 1 1 0 0 0 0 0 0 0 ... 0
3 0 0 0 0 0 0 0 0 0 0 0 ... 1
4 0 0 0 0 0 0 0 0 0 0 0 ... 0
5 0 1 0 01 0 0 0 0 0 0 ... 0
6 0 0 0 0 0 0 0 0 0 1 ... 0
7 0 0 0 0 0 0 0 0 0 0 ... 0
8 0 1 0 0 0 0 0 0 0 0 ... 0
9 0 0 0 1 0 10 0 0 0 ... 0
10 0 0 0 0 0 0 0 0 0 0 0 ... 0
11 0 0 0 0 0 0 0 0 1 0 ... 0
12 0 0 0 1 0 1 0 0 0 0 ... 0

Tomorrow very view we wentwhatwillwithwork0010 00 0 0 00

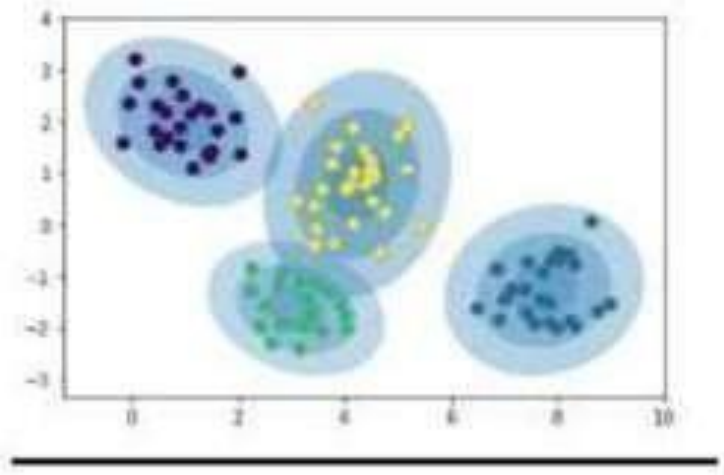
1	0	0	0	0	0 0	0	0
2	0	0	0	0	0 0	0	0
3	0	0	0	0	1 0	0	0
4	0	0	0	0	0 0	0	0
5	0	0	0	0	0 0	0	0
6	0	0	0	0	0 0	0	1
7	1	0	0	1	0 0	1	0
8	0	0	0	0	0 0	0	0

11.AIM: Apply EM algorithm to cluster a set of data stored in a. CSV file. Use the same data setfor clustering using k-Means algorithm. Compare the results of the set two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

Source Code:

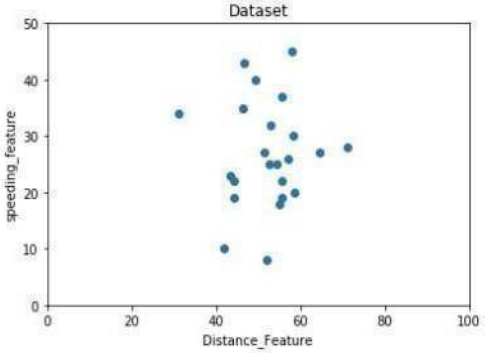
```
import numpy as np
import matplotlib.pyplot as plt
fromsk learn.datasets.samples_generator
import make _blobsX, y_ true = make_ blobs(n_ samples=100,centers
=4,Cluster_std=0.60,random_state=0)
X =X[:,::-1]

#flipaxesforbetterplotting

From sklearn. mixtureimportGaussianMixture
gmm=GaussianMixture(n_components=4).fit(X)lables = gmm.predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap="viridis");
probs = gmm.predict_proba(X)
print(probs[:5].round(3))
size=50*probs.max(1)**2#squareemphasizes
differencesplt.scatter(X[:,0],X[:,1],c=labels,cmap="viridis",s=size);

from matplotlib.patchesimportEllipse
def draw_ellipse(position, covariance, ax=None,
**kwargs);"""Drawanellipsewithagivenpositionandcovariance"""
Ax=axorplt.gca()
#Convertcovariancetoprincipalaxes
ifcovariance.shape ==(2,2):

U,s,Vt=np.linalg.svd(covariance)
Angle=np.degrees(np.arctan2(U[1,0],U[0,0]))Width,height= 2 * np.sqrt(s)
else:
angle=0
width,height=2*np.sqrt(covariance)

#DrawtheEllipse
fornsiginrange(1,4):
```

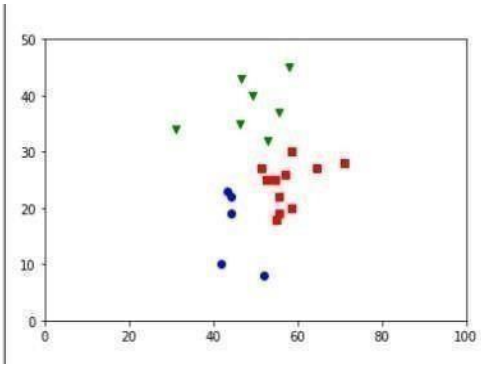
```
ax.add_patch(Ellipse(position,nsig*width,nsig*height,angle,**kwargs))

defplot_gmm(gmm,X,label=True,ax=None):ax= ax or plt.gca()
labels=gmm.fit(X).predict(X)iflabel:
ax.scatter(X[:,0],x[:,1],c=labels,s=40,cmap="viridis",zorder=2)

else:
ax.scatter(X[:,0],x[:,1],s=40,zorder=2)ax.axis(„equal")

w_factor=0.2/gmm.weights_.max()

forpos,covar,winzip(gmm.means_,gmm.covariances_,gmm.weights_):
draw_ellipse(pos,covar, alpha=w*w_factor)

gmm=GaussianMixture(n_components=4,random_state=42)plot_gmm(gmm,X)
gmm=GaussianMixture(n_components=4,covariance_type="full",random_state=42)
plot_gmm (gmm, X)
```

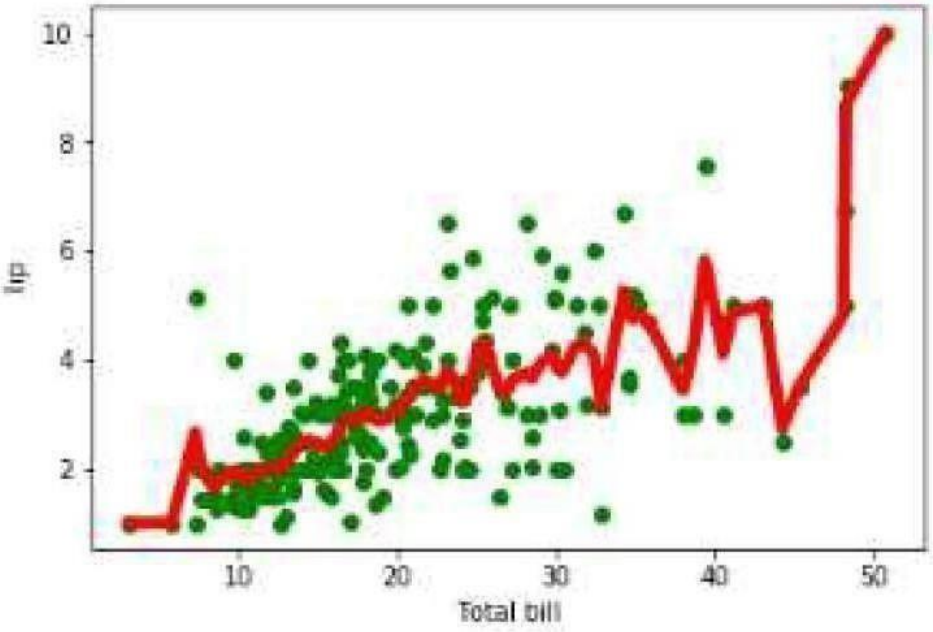


Output :

[[1,0, 0, 0]
[0,0,1,0]
[1,0,0,0]
[1,0,0,0]
[1,0, 0, 0]]


Dataset
speeding_feature
Distance_Feature
0
10
20
30
40
50
0
20
40
60
80
100


0
10
20
30
40
50
0
20
40
60
80
100


tip
Total bill
2
4
6
8
10
10
20
30
40
50

12.AIM: Exploratory Data Analysis for Classification using Pandas or Matplotlib

Source code:

```
Import pandas as pd
Import matplotlib. pyplot as plt
DF =pd. read_csv("https://raw.githubusercontent.com
Df =pd.read_csv("https://vincentarelbundock.github.io / Rdatasets / csv / car / Child.csv")
DF.describe()
/ fivethirtyeight / data / master / airline-safety / airline-safety.csv")
y =list(DF.population)
plt. Boxplot(y)
plt. show()
```

```
>>> DF.describe()
        Unnamed: 0     population           age         income     statusquo
count  2700.000000    2700.000000  2699.000000    2602.000000  2.683000e+03
mean   1350.500000  152222.222222    38.548722   33875.864719 -1.118151e-08
std     779.567188  102198.039602    14.756415   39502.867120  1.000186e+00
min       1.000000    3750.000000    18.000000    2500.000000 -1.803010e+00
25%     675.750000   25000.000000    26.000000    7500.000000 -1.002235e+00
50%    1350.500000  175000.000000    36.000000   15000.000000 -4.558000e-02
75%    2025.250000  250000.000000    49.000000   35000.000000  9.685750e-01
max    2700.000000  250000.000000    70.000000  200000.000000  2.048590e+00
```

```
DF["education"].value_counts()
DF.groupby(['education', 'vote']).mean()
From scipy.stats importf_oneway

# Sample data for three groups
group1 =[5, 7, 3, 4, 8]
group2 =[9, 12, 11, 13, 10]
group3 =[14, 16, 19, 17, 15]

# Perform ANOVA
f_statistic, p_value =f_oneway(group1, group2, group3)

# Print the results
print("F-statistic:", f_statistic)
print("p-value:", p_value)
```

Output:

```
>>> DF.groupby(['education', 'vote']).mean()
                Unnamed: 0     population         age         income  statusquo
education vote
P         A     1231.346154  141538.461538  40.653846  19489.795918  -0.207768
          N     1289.187970  148312.969925  43.018797  16650.763359  -0.926576
          U     1362.611486  129518.581081  46.091525  16783.216783   0.082548
          Y     1186.511848  107316.350711  44.902844  18469.512195   0.938229
PS        A     1363.750000  183476.562500  31.718750  48467.741935  -0.237501
          N     1374.111607  184492.187500  33.071429  63238.636364  -0.934572
          U     1562.019231  174447.115385  33.634615  58281.250000  -0.062994
          Y     1466.438462  179653.846154  35.469231  88252.032520   1.058944
S         A     1232.339806  161225.728155  32.029126  35176.767677  -0.165996
          N     1370.118388  175771.410579  32.438287  36071.428571  -0.885498
          U     1450.565401  175047.468354  34.489451  30325.112108   0.007525
          Y     1339.598071  144192.122186  35.993569  39975.328947   0.898503
```

13.AIM: Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. Youcan use Java/Python ML library classes/API

Theory:

A Bayesian network is a directed a cyclic graph in which each edge corresponds to a conditional dependency,

and each node corresponds to a unique random variable.

Bayesian network consists of two major parts: a directed a cyclic graph and a set of conditional probability distributions

- The directed acyclic graph is a setoff random variables represented by nodes.
- The conditional probability distribution of anode (random variable) is defined for every possible out come of the preceding causal node(s).

For illustration, consider the following example. Suppose we attempt to turn on our computer, but the computer does not start (observation/evidence). We would like to know which of the possible causes ofcomputer failure is more likely. In this simplified illustration, we assume only two possible causes of this misfortune: electricity failure and computermal function.

The corresponding directed acyclic graph is depicted in below figure.

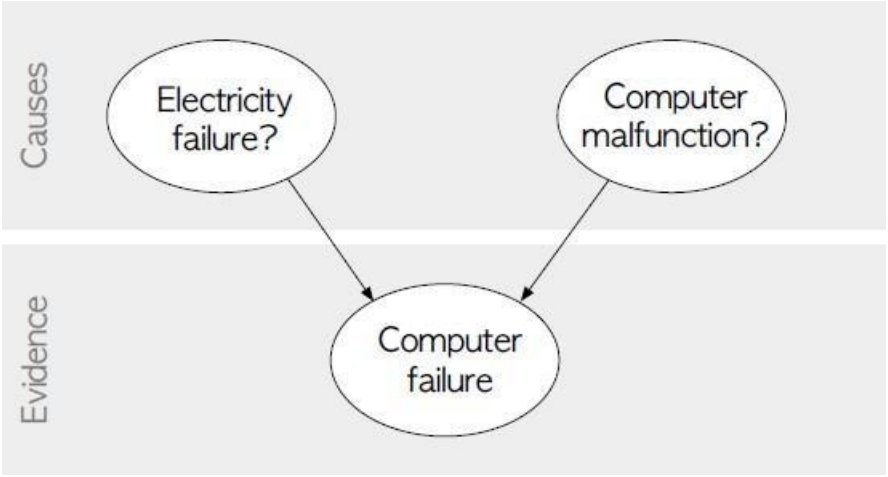


Fig: Directedacyclicgraphrepresentingtwoindependentpossiblecausesofacomputerfailure.

The go alis to calculate the posterior conditional probability distribution of each of the possible unobserved causes given the observed evidence, i.e. [Cause |Evidence].

DataSet:

Title: Heart Disease Databases

The Cleveland database contains 76 attributes, but all published experiments refer to using asubset of 14 of them. In particular, the Cleveland database is the only one that has been usedbyMLresearcherstothisdate.The"Heartdisease"fieldreferstothepresenceofheartdiseasein the patient. It isintegervaluedfrom0 (no presence) to4.

Database:	0	1	2			Total
Cleveland:	164	55	36	35	13	303

Attribute Information:

1. age: ageinyears
2. sex: sex (1 =male;0=female)
3. cp: chest pain type
   - Value1: typicalangina
   - Value2:atypicalangina
   - Value3:non-anginalpain
   - Value4:asymptomatic
4. trestbps: restingbloodpressure(inmmHgonadmissiontothehospital)
5. chol:serumcholestoralinmg/dl
6. fbs:(fastingblood sugar >120 mg/dl)(1=true; 0 =false)
7. restecg: restingelectrocardiographicresults
   - Value0: normal
   - Value1: havingST-Twaveabnormality (Twaveinversionsand/orSTelevationordepressionof>0.05mV)
   - Value2: showingprobableordefiniteleftventricularhypertrophybyEstes’criteria
8. thalach: maximum heartrateachieved
9. exang: exercise induced angina (1 =yes;0=no)
10. oldpeak=STdepressioninduced byexerciserelative torest
11. slope: the slope of thepeakexercise STsegment
    - Value 1: upsloping
    - Value 2: flat
    - Value3: down sloping
12. ca=number of majorvessels(0-3) colored byflourosopy

13. thal:3=normal;6=fixed defect; 7=reversable defect

Heart disease:

Itisintegervaluedfrom0(nopresence)to4.Diagnosisofheartdisease(angiographicdiseasestatus)

Some instance from the dataset:

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	Heartdisease
63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
62	0	4	140	268	0	2	160	0	3.6	3	2	3	3
60	1	4	130	206	0	2	132	1	2.4	2	2	7	4

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	Heartdisease
63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
62	0	4	140	268	0	2	160	0	3.6	3	2	3	3
60	1	4	130	206	0	2	132	1	2.4	2	2	7	4

Program:

```
import numpy as np
import csv
import pandas aspd
frompgmpy. modelsimportBayesianModel
from pgmpy.
estimatorsimportMaximumLikelihoodEstimatorfrompgmpy.inferenceimportVariableElimination

#readClevelandHeartDiseasedata heartDisease=pd.read_csv('heart.csv')
heartDisease=heartDisease. Replace (‘?’, np.nan)

#displaythedata
print('Few examples from the dataset are given below')
print(heartDisease.head())

#ModelBayesianNetwork Model=BayesianModel([('age','trestbps'), ('age','fbs'),
('sex','trestbps'), ('exang','trestbps'), ('trestbps','heartdise
ase'),('fbs','heartdisease'), ('heartdisease','restecg'),
('heartdisease','thalach'), ('heartdisease','chol')])

#LearningCPDsusingMaximumLikelihoodEstimators
print ('\n Learning CPD using Maximum likelihood estimators')model.fit(heartDisease,
estimator=MaximumLikelihoodEstimator)

#InferencingwithBayesianNetwork
print('\n Inferencing with Bayesian Network:')
HeartDisease_infer=VariableElimination(model)
```

#computingtheProbabilityofHeartDiseasegivenAge
print('\n 1. Probability of HeartDisease given
Age=30')q=HeartDisease_infer.query(variables=['heartdisease'],evidence
={'age':28})
print(q['heartdisease'])

#computing the Probability of HeartDisease given cholesterolprint('\n 2. Probability of HeartDisease given cholesterol=100')q=HeartDisease_infer.query(variables=['heartdisease'],evidence
={'chol':100})

print(q['heartdisease'])

Output:

Fewexamplesfromthedatasetaregivenbelow

	agesexcptrestbps		...slopecathalheartdisease0		63	1		1 145
		3	0	6				
1	67	4	160	...2	3	3		
2	67	4	120	...2	2	7		
3	37	3	130	...	0	3		
4	41	2	130	...	0	3		

[5rowsx14columns]

Learning CPD using Maximum likelihood

EstimatorsInferencingwithBayesianNetwork:

1. ProbabilityofHeartDisease given Age=28

heartdisease	phi(heartdisease)
heartdisease_0	0.6791
heartdisease_1	0.1212
heartdisease_2	0.0810
heartdisease_3	0.0939
heartdisease_4	0.0247

1. Probability of Heart Disease given cholesterol=100

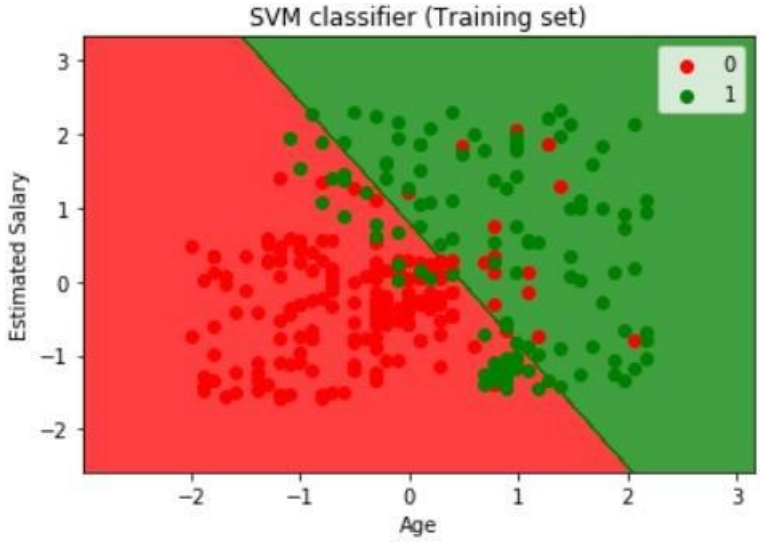
heartdisease	phi(heartdisease)
heartdisease_0	0.5400
heartdisease_1	0.1533
heartdisease_2	0.1303
heartdisease_3	0.1259
heartdisease_4	0.0506

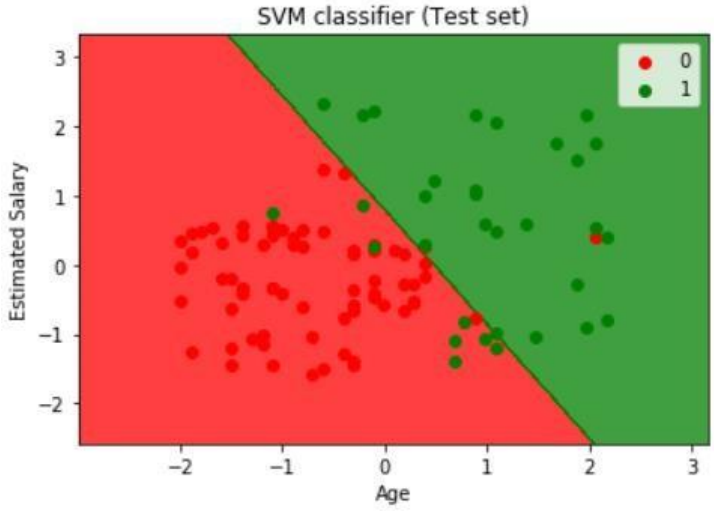
14.AIM: Write a program to Implement Support Vector Machines and Principle ComponentAnalysis.

Source code:

```
#Data Pre-processing Step
# importing libraries
import numpy as np
import matplotlib. pyplot as mtp
import pandas as pd
#importing datasets
data_set= pd.read_csv('user_data.csv')
#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values
# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
#feature Scaling
from sklearn. preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
#Predicting the test set result
y_pred= classifier.predict(x_test)
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1. shape ),


alpha = 0.75, cmap = ListedColormap(('red', 'green')))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm. unique(y_set)):
   mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
      c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('SVM classifier (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')mtp.legend() bmtp.show()
```

Output:

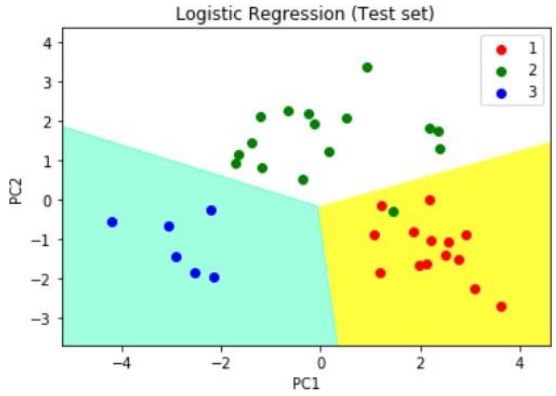

SVM classifier (Training set)
Estimated Salary
Age
0
1
3
2
1
0
−1
−2
−2
−1
0
1
2
3


SVM classifier (Test set)
Estimated Salary
Age
0
1
3
2
1
0
−1
−2
−2
−1
0
1
2
3

15.AIM: Write a program to Implement Principle Component Analysis.Source code:

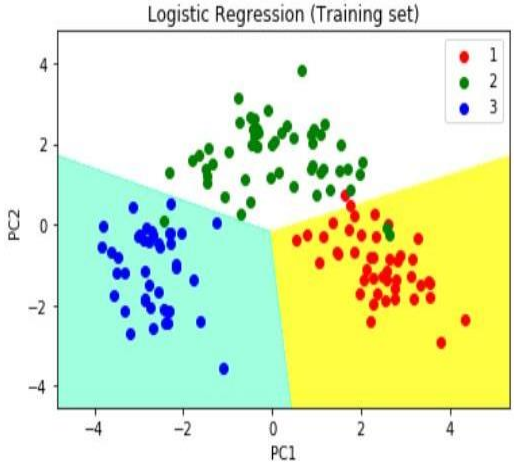
```
# importing required libraries
import NumPy as np
import matplotlib. pyplot as plt
importpandas as pd
# importing or loading the dataset
dataset =pd.read_csv('wine.csv')
 # distributing the dataset into two components X and Y
X =dataset. iloc[:, 0:13].values
y =dataset.iloc[:, 13].values
# Splitting the X and Y into the
# Training set and Testing set
From sklearn. model selection importtrain_test_split
 X_train, X_test, y_train, y_test =train_test_split(X, y, test_size =0.2, random_state =0)
# Applying PCA function on training
# and testing set of X component
From sklearn. Decomposition import CA
 pca =PCA(n_components =2)
 X_train =pca.fit_transform(X_train)
X_test =pca.transform(X_test)
 explained_variance =pca.explained_variance_ratio_

# Predicting the training set
# result through scatter plot
From matplotlib. colors importListedColormap
 X_set, y_set =X_train, y_train
X1, X2 =np.meshgrid(np.arange(start =X_set[:, 0].min() -1,
                    stop =X_set[:, 0].max() +1, step =0.01),
                    np.arange(start =X_set[:, 1].min() -1,
                    stop =X_set[:, 1].max() +1, step =0.01))


plt. contourf(X1, X2, classifier. predict(np.array([X1.ravel(),
            X2.ravel()]).T).reshape(X1.shape), alpha =0.75,
            cmap =ListedColormap(('yellow', 'white', 'aquamarine')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
 fori, j innumerate (np. unique(y_set)):
```

```
    plt. scatter (X_set[y_set ==j, 0], X_set[y_set ==j, 1],
                c =ListedColormap(('red', 'green', 'blue'))(i), label =j)
 plt.title('Logistic Regression (Training set)')
plt.xlabel('PC1')    # for Xlabel
```




```
plt.ylabel('PC2')     # for Ylabel
plt.legend()
```

Output:




```
# to show legend
```